VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

# Absolvování individuální odborné praxe

# Individual Professional Practice in the Company

2016 Miroslav Babral

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

# Zadání bakalářské práce

Student: **Miroslav Babral**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: Railsformers s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Martin Němec, Ph.D.**

Konzultant bakalářské práce: Ing. Kateřina Snídalová

Datum zadání: 01.09.2015

Datum odevzdání: 29.04.2016

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky ID16 OSTRAVA

doc. Dr. Ing. Eduard Sojka
*vedoucí katedry*

prof. RNDr. Václav Snášel, CSc.
*děkan fakulty*

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 6.4. 2016 ..... Balnal .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.


V Ostravě 6.4. 2016

Rád bych na tomto místě poděkoval především mé rodině, že mě vždy plně podporovala a umožnila mi mé studium. Dále bych chtěl poděkovat svému vedoucímu bakalářské práce panu Ing. Martinu Němci Ph.D, který mi byl vždy ochoten poradit a měl se mnou trpělivost. Dále bych chtěl poděkovat mému konzultantu z firmy, Ing. Kateřině Snídalové, za všechnu pomoc, kterou mi během mé bakalářské praxe poskytla.

**Abstrakt**

Toto zpracování bakalářské práce se bude zabývat mou bakalářskou praxí u ostravské programátorské firmy Railsformers s.r.o. Tato firma se zabývá vytvářením především webových aplikací pro své zákazníky ve webovém frameworku Ruby on Rails jazyka Ruby včetně jejich následného spravování. V této firmě jsem se zabýval nejprve s osvojením základních, ale později i pokročilejších programovacích technik v Ruby on Rails pod vedením svého přiděleného konzultanta z Railsformers s.r.o. a následným programováním webových aplikací právě pod Ruby on Rails. V této práci se pokusím detailně popsat průběh své bakalářské praxe. Včetně úkolů, které mi byly zadány, nových věcí, které jsem se v rámci této praxe naučil a znalostí, které jsem získal při mém studiu informatiky na Vysoké škole Báňské v Ostravě a následně je zužitkoval při vykonávání této bakalářské praxe.

**Klíčová slova**: bakalářská práce, bakalářská praxe, Ruby on Rails, Ruby, webový framework

**Abstract**

This written concept of bachelor thesis is about my bachelor practice at Ostrava's programming company Railsformers s.r.o. The company creates mostly a web applications for their customers in Ruby based web framework Ruby on Rails with subsequent administration. First things first, I have learned basic and later more intermediate programming techniques in Ruby on Rails under my assigned leading consultant from Railsformers s.r.o. and subsequent programming of web applications under Ruby on Rails framework. I am going to try to detaily describe progress of my bachelor practice. Including my tasks, which were assigned to me, new things I have learned during this practice and knowledge I have got during my study of IT on Technical University in Ostrava so far and subsequent application of it during my bachelor practice.

**Key Words**: bachelor thesis, bachelor practice, Ruby on Rails, Ruby, web framework

# Obsah

**Seznam použitých zkratek a symbolů** **9**

**Seznam obrázků** **10**

**Seznam výpisů zdrojového kódu** **11**

**1 Úvod** **12**

**2 O firmě** **13**

**3 Ruby on Rails** **14**
3.1 Jazyk Ruby . . . 14
3.2 Ruby on Rails . . . 14
3.3 Uplatnění v Ruby on Rails . . . 14
3.4 Ruby on Rails dnes . . . 14
3.5 Gemy . . . 15

**4 Použité technologie** **16**
4.1 Ubuntu . . . 16
4.2 MySQL . . . 16
4.3 Webová aplikace Projektove.cz . . . 16
4.4 GIT . . . 16
4.5 Bootstrap . . . 17
4.6 Architektura, návrhové vzory a metodiky vývoje . . . 17

**5 Praktická část bakalářské praxe** **19**
5.1 Časový harmonogram bakalářské praxe . . . 19
5.2 Seznámení se s prostředím firmy . . . 19
5.3 Blogovací webová aplikace . . . 19
5.4 Prostudování gemů . . . 19

**6 Inzertní webová aplikace** **25**
6.1 Zadání práce . . . 25
6.2 Moje práce . . . 25
6.3 Zhodnocení práce . . . 35

**7 Webová aplikace bankovních importů** **36**
7.1 Zadání práce . . . 36
7.2 Moje práce . . . 36

7.3 Zhodnocení práce . . . . . 45

**8 Závěr** **46**

**Literatura** **47**

# Seznam použitých zkratek a symbolů

| | | |
|---|---|---|
| RoR | – | Ruby on Rails |
| HTML | – | Hyper Text Markup Language[29] |
| Haml | – | HTML abstraction markup language[25] |
| MVC | – | Model View Controller[30] |
| SQLite | – | Typ databázového systému |
| MySQL | – | Typ databázového systému |
| SQL | – | Structured Query Language[31] |
| ERB | – | Embedded Ruby, HTML s podporou Ruby kodu uvnitř, používá se v view části aplikace[32] |
| XML | – | Extensible Markup Language, slouží k ukladání informací[33] |

# Seznam obrázků

1 Vodopádový model . . . . . 18
2 Relační model databáze pro inzertní aplikaci . . . . . 26
3 Relační model databáze pro aplikaci bankovní importy . . . . . 37

# Seznam výpisů zdrojového kódu


1 Ukázka formuláře využívajícího Simple form z inzertní aplikace . . . . . 19
2 Výsledný HTML Simple form z Výpisu č.1 . . . . . 20
3 Ukázka Haml z inzertní aplikace . . . . . 21
4 Ekvivaletní zápis Výpisu 5. v HTML . . . . . 22
5 Nastavení stránkování pro inzeráty . . . . . 22
6 Test modelu Contact v RSpec . . . . . 22
7 Validace modelu Category . . . . . 27
8 Validace modelu Advertisement . . . . . 27
9 Zmenšení obrázku třída ImageUploader . . . . . 29
10 Carrierwave složka pro ukládání obrázků . . . . . 29
11 Validace modelu Contact . . . . . 29
12 Ukázka z českého lokalizačního souboru . . . . . 30
13 Prodání inzerátu a jeho následné možné vrácení do inzerce s překladem . . . . . 30
14 Vytvoření uživatele při vytvoření inzerátu neregistrovaným uživatelem . . . . . 31
15 Vytvoření názvu konverzace při vytvoření nové zprávy . . . . . 32
16 Ověření existence konverzace v systému . . . . . 32
17 Úprava na dynamické přídávání až deseti fotek . . . . . 33
18 Část view obsahující přesměrování na napsání odpovědi na inzerát . . . . . 34
19 Příkaz do příkazové řádky pro vytvoření migrace . . . . . 34
20 Obsah vytvořené migrace v aplikaci . . . . . 35
21 Příkaz pro změnu databáze pomocí migrace . . . . . 35
22 Nastavení základních pravomocí jednotlivých rolí v tříde Ability . . . . . 38
23 Při vytvoření uživatele implicitní role user . . . . . 39
24 Omezení pravomocí v aplikaci pomocí roli ve views . . . . . 39
25 Vytvoření instance typu Profile po vytvoření instance typu User . . . . . 40
26 Počet všech uživatelů role admin v controleru . . . . . 40
27 Výpis všech uživatelů v systému včetně možnosti změnit roli ve views . . . . . 41
28 Formulář pro vytvoření XML mezi dvěma datumy . . . . . 42
29 Export do XML mezi dvěma datumy v metodě index v controlleru Transaction . 42
30 Výpis všech přiložených souborů u bankovního účtu ve view index . . . . . 44
31 Metoda download v controlleru Attachment umožňující stáhnutí přílohy . . . . . 44

# 1 Úvod

Tato práce se bude zabývat mou bakalářskou praxí, kterou jsem vykonával v počítačové firmě Railsformers s.r.o. sídlící v Ostravě. V této firmě jsem vykonával praxi jako Ruby on Rails programátor. Během této praxe na mě dohlížel mi přidělený konzultant (programátor) z firmy Railsformers s.r.o.

Nejprve jsem dostal za úkol nastavení všech prostředí nutných pro vývoj v Ruby on Rails. Poté jsem musel nastudovat určitou část dokumentace. Po nastudování dokumentace jsem dostal svůj první úkol, abych využil své nově nabyté znalosti. Můj první úkol bylo naprogramovat blogovou webovou aplikaci.

Po dokončení tohoto projektu, byl můj úkol v týmu s ještě jedním stážistou navrhnout a naprogramovat inzertní webovou aplikaci.

Mým posledním úkolem bylo navrhnout a naprogramovat webovou aplikace, která by měla zpracovávat bankovní importy zákazníků.

Cíle mé práce jsou naučení se nových programovacích znalostí, získání důležitých zkušenosti z firemního prostředí a naučit se spolupráci s více lidmi v rámci programování na větších firemních projektech.

Na praxi jsem docházel během letního a zimního semestru po dobu 50-ti dnů po běžné pracovní době 8-mi hodin.

Tato práce je rozdělena do následujících kapitol:

- Ruby on Rails - popis webového frameworku Ruby on Rails
- Praktická část bakalářské praxe - můj začátek bakalářské praxe ve firmě Railsformers s.r.o.
- Inzertní webová aplikace - popis zajímavých částí z mé práce na projektu inzertní webové aplikace
- Webová aplikace bankovních importů - popis zajímavých částí z mé práce na projektu webové aplikace bankovních importů
- Závěr - mé závěrečné zhodnocení mé bakalářské praxe

# 2 O firmě

Firma, ve které jsem vykonával svou bakalářskou praxi je Railsformers s.r.o. Jedná se o ostravskou počítačovou firmu. Podle počtu zaměstnanců patří do kategorie mikropodniků.[1]

Zabývá se především vývojem webových aplikací v Ruby on Rails včetně SEO optimalizace pro webové vyhledávače jako například Google, Seznam, apod. Vytváří také mobilní aplikace spojené s webovými aplikacemi svými nebo třetích stran.

Mezi softwarové produkty této firmy patří například:

1. sMoneybox - webová i mobilní aplikace pro přehled svých útrat[2]
2. sÚčto - webová aplikace pro vytváření faktur online[3]
3. Hedurio - nejvýznamnější produkt této firmy. Jedná se online informační systém pro bezpečnostní agentury[4]
4. Rainbow tours - webová stránka pro cestovní společnost Rainbow tours s možností výběru dovolené[5]

Firma byla založena v roce 2010 panem Ing. Jiřím Kubicou a je stále pod jeho vedením.[6]

# 3 Ruby on Rails

V této části bych se chtěl věnovat základním informacím o webovém frameworku Ruby on Rails.

## 3.1 Jazyk Ruby

Ruby je jádrem webového frameworku Ruby on Rails. Ruby byl vytvořen japonským programátorem Yukihiro Matsumotem v roce 1995.

Jedná se o skriptovací programovací jazyk, který podporuje objektivně orientované programování a dynamické typování. Jeho cílem byl důraz na lepší čitelnost pro programátora a pohodlnější syntaxi, například není nutné psát středníky za každým příkazem nebo není nutno kód kompilovat, stačí pouze kód spustit v překladači.[7]

## 3.2 Ruby on Rails

Ruby on Rails byl vytvořen Davidem Heinemeier Hanssonem, který jej zveřejnil roku 2004 a v roce 2005 z něj udělal opensource. Tento webový framework využívá MVC architekturu. Jako databázi podporuje např. SQLite nebo MySQL. Mezi jeho uživatele patřili například Apple nebo Twitter.

Ruby on Rails je rozvíjen velkou komunitou programátorů, takže se neustále vyvíjí. Proto pokud je nalezena nějaká chyba, bývá většinou poměrně rychle opravena. Tito programátoři postupem času rozšířili tento nový webový framework o spoustu důležitých a dnes již nepostradatelných funkcí. Mimo jádro frameworku Ruby on Rails je možno přidat další knihovny, které jsou vytvářeny opět rozsáhlou komunitou programátorů, nazývané gems. Tyto knihovny slouží k usnadnění práce programátorů. Umožňují jim například přidání autorizačního systému ( Devise, omniauth ), využití frontendových frameworků (Bootstrap, Foundation) atd.[8, 9]

## 3.3 Uplatnění v Ruby on Rails

V České republice není Ruby on Rails příliš obvyklým webovým frameworkem v porovnání například s jazykem J2EE nebo ASP.NET. Usoudil jsem tak z poptávky na webových portálech zabývajících se inzercí volných pracovních míst. Zřejmě kvůli tomu, že se jedná o poměrně nový webový framework. Do širšího povědomí se dostal až kolem roku 2010. Ovšem jedná se o 10. nejpoužívanější webový framework na světě s více než 860.000 vytvořených webových stránek na celém internetu k datu 25.1.2016.[10]

## 3.4 Ruby on Rails dnes

V dnešní době se Ruby on Rails se využívá především v menších firmách. Větší firmy jako například Twitter od něho nedávno ustoupili. Ovšem Ruby on Rails se neustále vyvíjí dopředu. Jeho budoucnost je zřejmě v menších firmách. Jeho výhoda spočívá ve velmi rychlém vývoji

a čitelnosti kódu. Zkušený programátor je schopen vytvořit blogovou webovou aplikaci během několika hodin až dnů.

## 3.5 Gemy

Gem je název pro knihovnu tříd, která rozšiřuje funkčnost naší aplikaci o již naprogramované funkce. Tyto gemy pochází buď přímo od vývojářů Ruby on Rails nebo od běžných programátorů. Ulehčují programátorovi práci. Gem, který chce programátor použít musí být nainstalován přímo do aplikace přes příkazovou konzoli. Všechny nainstalované gemy jsou vypsány v souboru Gemfile.

Umožňují přidání autentizačního systému (Devise, omniauth), frontendových frameworků (Bootstrap, Foundation), alternativního stylovacího jazyku (Sass, Less), alternativního značkovacího jazyka (Haml), vyhledávání na stránce (Ransack) nebo upload obrázkových souborů (CarrierWave, Paperclip).[11]

# 4 Použité technologie

Při své bakalářské praxi ve firmě jsem používal i jiné technologie než jen Ruby on Rails nebo čistého Ruby. V této části popíši další použité technologie během mé bakalářské praxe.

## 4.1 Ubuntu

Ubuntu je linuxový operační systém. Jeho první verze vyšla v roce 2004 a od té doby vychází nová verze každého půl roku. Využívá debianovou architekturu. Jeho nejpřednější vlastností je vysoká úroveň zabezpečení.[12, 13, 14]

Nejprve jsem používal operační systém Windows 8. Nicméně Ruby on Rails plně nepodporuje operační systém Windows. Proto jsem kvůli nastalým problémům přešel na operační systém Ubuntu.

## 4.2 MySQL

MySQL je opensourcový relační databázový systém. Jedná se o druhý nejpoužívanější databázový systém na světě. Pro své dotazy využívá upravenou verzi jazyka SQL.[15, 16]

Pro každou webovou aplikaci bylo nutné navrhnout databázi a následně ji vytvořit. Nejprve jsem vyzkoušel u svých webových aplikací databáze SQLite. Kvůli tomu, že ve se ve firmě používají databázové systémy od MySQL, přešel jsem k MySQL. Nicméně jsem žádný rozdíl ve funkčnosti nenašel.

## 4.3 Webová aplikace Projektove.cz

Webová aplikace Projektově.cz slouží k organizaci práce zaměstnanců ve firmě. Umožňuje vedoucím pracovníkům, aby udělovali úkoly svým podřízeným a kontrolovali jejich postup práce. Dále umožňuje vzájemnou komunikaci jak mezi nadřízenými a podřízenými, tak mezi kolegy spolupracujících na stejných projektech.

Na webové aplikaci Projektove.cz jsme měli zadané projekty od našeho konzultanta. Vytvářeli jsme si podúkoly v těchto zadaných projektech, a pokud byly dokončené, označili jsme je za hotové.[17]

## 4.4 GIT

GIT je název pro skupinu softwarů, které umožňují programátorům spolupracujících na společném projektu, aby si mezi sebou mohli synchronizovat práci, kterou každý z nich provedl. Tento software funguje tak, že má na serveru uloženou verzi projektu, kterou si nejprve stáhnou všichni programátoři. Jakmile někdo z těchto programátorů dokončí svůj úkol, nechá GIT spojit nově upravenou část projektu se starší verzí nebo už upravenou verzí projektu uloženou na serveru. GIT neumí řešit sám kolize, tj. pokud dva programátoři upraví stejnou část projektu různě, poté musí člověk sám určit, jak bude výsledná podoba této části projektu vypadat.

Projekty byly často rozsáhle, proto jsme spolupracovali s dalšími stážisty v týmech. Každý z nás plnil své úkoly. Abychom si mohli synchronizovat naše provedené změny v projektu, využívali jsme GITu konkrétně Github.com.[18]

## 4.5 Bootstrap

Bootstrap je volně stažitelný frontendový framework. Byl vyvinut lidmi ze společnosti Twitter. Jeho cílem bylo umožnění zjednodušení nastylování všech komponent webové stránky (Form, Button) a přidává své vlastní komponenty (Popup, Dropdown). Pro nastylování stránek využívá jazyků CSS a jQuery.[19]

K nastylování webových aplikací jsem používal frontendového frameworku Bootstrap.

## 4.6 Architektura, návrhové vzory a metodiky vývoje

Ruby on Rails využívá návrhového vzoru MVC a Active Records. Jako metodiku vývoje jsme ve firmě využívali vodopádového modelu.

### 4.6.1 MVC

MVC je architektura softwarového díla, která rozděluje aplikaci na tři hlavní části:[30]

- Model obsahující informace z databáze, se kterou je spojen. Provádí také změny dat v databázi. Uživatel aplikace nemá do této části aplikace přístup.
- View zobrazuje informace uživateli aplikace v přehledné podobě a které je schopen porozumět.
- Controller obstarává veškerou logiku aplikace. Může reagovat na události a také měnit View i Model. Uživatel aplikace nemá do této části aplikace přístup.

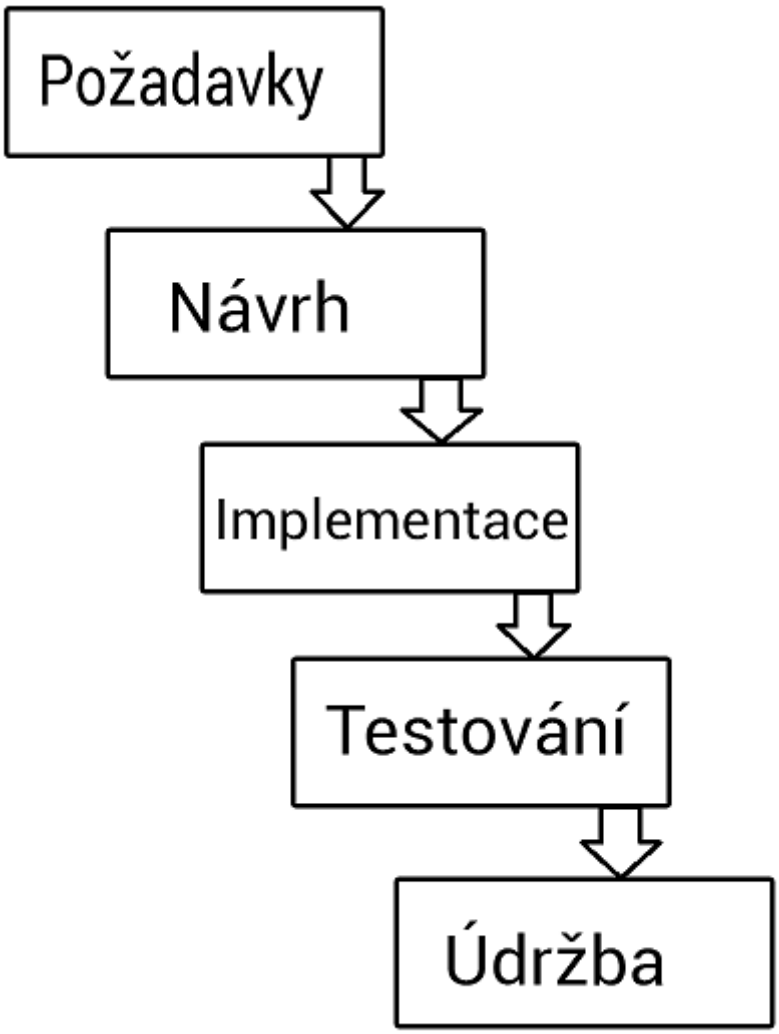
### 4.6.2 Active Records

Jedná se návrhový vzor, který umožňuje, aby každá instance třídy, například Person, byla jedním řádkem v tabulce Person v databázi. Umožňuje nám přes tento způsob data ukládat nebo upravovat. Dále umožňuje Active Records hledat nebo mazat záznam například v tabulce Person přes objekt Person. Objekt Person zastupuje celou tabulku Person v databázi.[21]

### 4.6.3 Vodopádový model

Vodopádový model je metodika vývoje softwarového díla, kterou jsem využíval při mé praxi ve firmě. S touto metodikou vývoje jsem byl již seznámen v předmětu Úvodu do softwarového inženýrství(SWI).

Nejprve jsme vždy analyzovali požadavky. Podle analýzy požadavků jsme vytvořili návrh, jak by měla webová aplikace vypadat. Podle návrhu jsme implementovali. Dále jsme pro aplikaci napsali testy. Nakonec jsme diskutovali s naším vedoucím o případné změny.[22]




Obrázek 1: Vodopádový model

# 5 Praktická část bakalářské praxe

V této části se budu věnovat svým začátkům ve firmě.

## 5.1 Časový harmonogram bakalářské praxe

1. Seznámení se s prostředím firmy, nastudovaní potřebných materiálů a jejich odzkoušení
2. Vývoj webové inzertní aplikace
3. Vývoj webové aplikace pro bankovní importy

## 5.2 Seznámení se s prostředím firmy

První den mé praxe spočíval v nainstalování všech potřebných komponent jazyka Ruby a Ruby on Rails na Windows. Nicméně Ruby on Rails plně nepodporuje Windows, proto jsem byl nucen přejít na Ubuntu.

Pokračoval jsem tedy nainstalováním Ubuntu a následně nainstalování nezbytných komponent ke správné funkčnosti jazyka Ruby i jeho frameworku Ruby on Rails na Ubuntu.

Prostudoval jsem vybranou dokumentaci mým konzultantem k Ruby on Rails z oficiálních zdrojů a jejich následné vyzkoušení.

## 5.3 Blogovací webová aplikace

Mým dalším úkolem bylo vytvoření blogovací webové aplikace. Nejprve jsem si musel vytvořit návrh databáze v databázovém systému SQLite.

Jednalo se o jednoduchý úvodní projekt. V této webové aplikaci měla být úvodní stránka, kde byly jednotlivé zprávy (Post) a byla zde také možnost přidat další zprávy. Po zobrazení detailu této zprávy měla být možnost, aby anonymní uživatel mohl přidat komentář.

## 5.4 Prostudování gemů

Dále jsem dostal za úkol prostudovat dokumentaci a zhlédnout výuková videa k gemům. Po dostudování gemů jsem si měl nastavit komunikaci mého počítače s firemním GIT kanálem na Githubu.

### 5.4.1 Simple form

Simple form slouží k usnadnění zápisu HTML formulářů. Funguje způsobem, že již nepotřebujeme využívat u formulářů HTML jazyk, ale napíšeme pouze zkrácenější verzi v Ruby on Rails. Ten se při spuštění webové aplikace přeloží jako HTML značky.

```
<%= simple_form_for(@contact) do |f| %>
  <%= f.input :firstname, label: t('Firstname') %>
```

```
    <%= f.input :surname, label: t('Surname') %>
    <%= f.input :phonenumber, label: t('Phonenumber') %>
    <%= f.input :city, label: t('City') %>
    <%= f.input :street, label: t('Street') %>
    <%= f.input :zipcode, label: t('Zipcode') %>
    <%= f.submit t('SubmitButton') %>
<% end %>
```

Výpis 1: Ukázka formuláře využívajícího Simple form z inzertní aplikace

V příkladu výše vytvoříme HTML formulář. Ruby on Rails nepoužívá čistého HTML souboru, ale má svou upravenou verzi, ve které funguje jazyk Ruby. Koncovka tohoto upraveného HTML souboru má tvar nazev.html.erb.

Funguje to tak, že instanční proměnná contact se uloží do lokální proměnné f. Pomocí proměnné f přistupujeme k jednotlivým atributům (firstname, surname, ...) proměnné contact. Na konci se vytvoří Button s metodou submit, která odešle formulář. Každý řádek f.inputu obsahuje také text, který bude napsaný v Labelu k Inputu. Klíčové slovo **t** znamená, že dojde k překladu z lokalizačního souboru. U **t**('Firstname') se najde v lokalizačním souboru prvek s názvem Firstname a jeho hodnota se napíše jako text do Labelu.

```
<form method="post">
    <label for="user_contact_attributes_firstname">Jmeno</label>
    <input id="user_contact_attributes_firstname" type="text" name="user[
        contact_attributes][firstname]" value=""></input>

    <label for="user_contact_attributes_surname">Prijmeni</label>
    <input id="user_contact_attributes_surname" type="text" name="user[
        contact_attributes][surname]" value=""></input>

    <label for="user_contact_attributes_phonenumber">Telefon</label>
    <input id="user_contact_attributes_phonenumber" type="text" name="user[
        contact_attributes][phonenumber]" value=""></input>

    <label for="user_contact_attributes_city">Mesto</label>
    <input id="user_contact_attributes_city" type="text" name="user[
        contact_attributes][city]" value=""></input>

    <label for="user_contact_attributes_street">Ulice</label>
    <input id="user_contact_attributes_street" type="text" name="user[
        contact_attributes][street]" value=""></input>
```

```
    <label for="user_contact_attributes_zipcode">Smerovacc cislo</label>
    <input id="user_contact_attributes_zipcode" type="text" name="user[
        contact_attributes][zipcode]" value=""></input>

    <input type="submit" value="Submit">
</form>
```

Výpis 2: Výsledný HTML Simple form z Výpisu č.1

Více informací je možno najít zde [23].

### 5.4.2 Devise

Devise slouží k vytvoření autentizačního systému ve webové aplikaci.

Po nainstalování gemu Devise se v databázi vytvoří tabulka User, která obsahuje klasické atributy jako email a heslo (zašifrované). Mimo tyto dva atributy obsahuje i další informativní atributy jako například datum a čas vytvoření. Devise nebrání upravit si tabulku dle svého uvážení.

Vytvoří se nám také views, které obsahují formuláře zabývající se vším ohledně registrace a přihlášení (ztracené heslo, změna hesla, atd.). Potom je nutné upravit části kódu, aby spojil Devise s právě vytvářenou aplikací.

Více informací je možno najít zde [24].

### 5.4.3 Haml

Haml je značkovací jazyk, který slouží jako náhrada pro HTML. Jeho proklamovanou výhodou je větší přehlednost a snadnější zápis oproti HTML. Jak lze vidět na úryvku kódu z projektu inzertní aplikace, na kterém jsem pracoval níže.

Více informací o Hamlu lze najít na oficiální stránce [25].

```
%h1= t('AdvertisementList')
%table
    %thead
        %th= t('Description')
        %th= t('Price')
        %th= t('PostDate')
        %th= t('EndDate')
    %tbody
```

Výpis 3: Ukázka Haml z inzertní aplikace

```
<h1><%= t('AdvertisementList') %></h1>
<table>
  <thead>
    <th><%= t('Description') %></th>
    <th><%= t('Price') %></th>
    <th><%= t('PostDate') %></th>
    <th><%= t('EndDate') %></th>
  </thead>
  <tbody>
  </tbody>
</table>
```

Výpis 4: Ekvivaletní zápis Výpisu 5. v HTML

### 5.4.4 Kaminari

Kaminari je gemem pro Ruby on Rails, který umožňuje jednodušším způsobem implementovat stránkování výpisů záznamů. V praxi jsem je například využil u inzertní aplikace, když jsem vypisoval právě dostupné inzeráty. Úryvek aplikace Kaminari z inzertní aplikace je níže.

```
@advertisements = Advertisement.where("end_date >= ? AND sold=false", Date.
    today.midnight).order(create_date: :DESC).page(params[:page]).per(10)
```

Výpis 5: Nastavení stránkování pro inzeráty

Příklad výše se odehrává v controlleru Advertisement. Pomocí modelu Advertisement načteme do takové inzeráty, které mají datum platnosti do dnešní půlnoci a nebyly doposud prodány. Tyto inzeráty seřadíme od nejnovějších. Pomocí Kaminari na konci řádku určíme, že všechny inzeráty budou stránkovány a to tak, že na jedné stránce bude nejvýše 10 záznamu z modelu Advertisement. Všechny záznamy se zde ukládají do instanční kolekce, která se poté vypisuje v patřičném view.

Více informací lze najít na [26].

### 5.4.5 RSpec

RSpec je framework, který slouží k testování aplikace. V praxi jsem tento framework používal především pro testování modelů. Úryvek testování z inzertní aplikace je níže.

```
describe "relation" do
  it {should have_one :user}
end
```

```
describe "validation" do
   it {should validate_presence_of :firstname}
   it {should validate_presence_of :surname}
   it {should validate_presence_of :city}
   it {should validate_presence_of :street}
   it {should validate_presence_of :zipcode}

   describe "is valid: firstname has more than 2 & has less than 50 letters"
        do
      it { should ensure_length_of(:firstname).is_at_least(2) }
      it { should ensure_length_of(:firstname).is_at_most(50) }
   end

   describe "is valid: surname has more than 2 & has less than 50 letters"
       do
      it { should ensure_length_of(:surname).is_at_least(2) }
      it { should ensure_length_of(:surname).is_at_most(50) }
   end

   describe "is valid: city has more than 2 & has less than 50 letters" do
      it { should ensure_length_of(:city).is_at_least(2) }
      it { should ensure_length_of(:city).is_at_most(50) }
   end

   describe "is valid: street has more than 2 & has less than 50 letters" do
      it { should ensure_length_of(:street).is_at_least(2) }
      it { should ensure_length_of(:street).is_at_most(50) }
   end

   describe "is valid: zipcode is integer" do
      it { should validate_numericality_of(:zipcode).only_integer }
   end
end
```

Výpis 6: Test modelu Contact v RSpec

V úryvku výše testuji nejprve předepsanou vazbu mezi Contact a User, kdy jeden User má k sobě přiřazenou jednu instanci Contact v oblasti **describe "relation"**. V oblasti **describe "validation"** nejprve testuji povinnou přítomnost všech atributů modelu Contact. Pokračuji otestováním zda atributy mají povolenou délku v rozpětí 2 až 50 znaků. A poslední část se

věnuje ověření zda atribut zipcode povoluje pouze celočíselné hodnoty konkrétně typu integer.

Po dopsání kódu je nutné ještě spustit test v příkazové řádce, který odhalí případné chyby. Více informací lze najít na [27, 28].

# 6 Inzertní webová aplikace

V rozmezí těchto dní jsem byl přiřazen s dalším stážistou do týmu, později k nám do týmu přidali další dva stážisty. Dostali jsme jako úkol větší projekt. Měli jsme naprogramovat inzertní webovou aplikaci.

## 6.1 Zadání práce

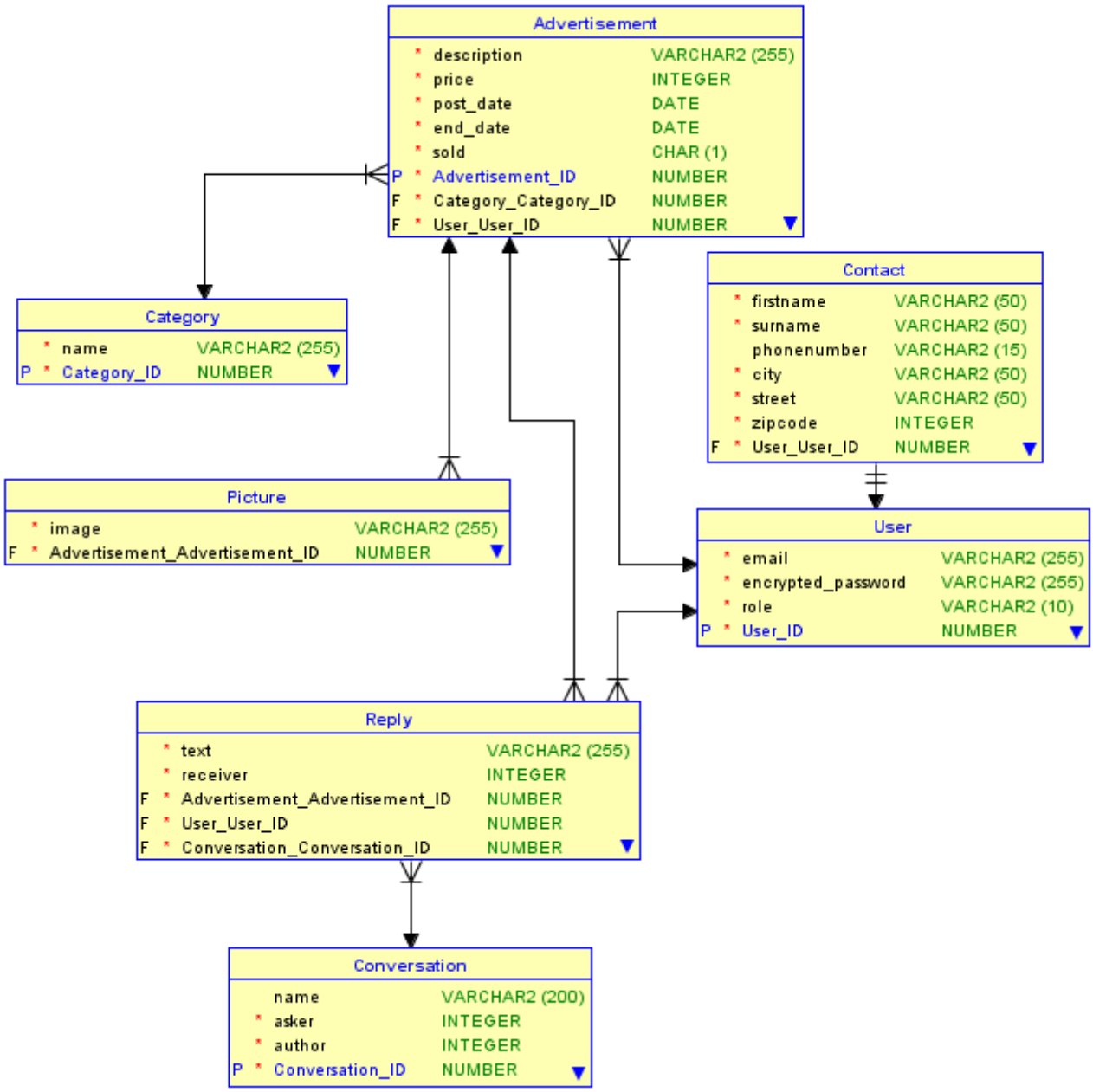
Inzertní aplikace měla obsahovat inzeráty organizované v kategoriích. Inzeráty se měli na stránce zobrazovat po stránkách. Ve kterých mohl uživatel filtrovat inzeráty podle rozpětí ceny nebo podle hledaných výrazů. Dále měl mít systém implementovaný autorizační a autentizační systém s více rolemi. Role měli být administrátor, registrovaný uživatel a neregistrovaný uživatel. Každá z těchto rolí měla jiné práva přístupu na stránce a rozdílné pravomoci. K inzerátu mělo jít přidat až deset obrázkových souboru, jejichž velikost se zmenšila z důvodu úspory místa. Aplikace měla mít dvě jazykové lokalizace - českou a anglickou. Systém měl také obsahovat komunikační systém na bázi zasílaní zpráv mezi uživateli. Aplikace měla také obsahovat sdílející tlačítka pro sociální sítě Google+, Twitter a Facebook. Nastylovaní aplikace mělo být provedeno v Bootstrapu. Nakonec měla být celá inzertní aplikace řádně otestována pomocí testů, který Ruby on Rails obsahuje přes gem Rspec.

## 6.2 Moje práce

V této části se budu zabývat výhradně částmi, které jsem v projektu inzertní webové aplikace zpracoval.

### 6.2.1 Analýza zadání projektu

Nejdříve jsme si s mým kolegou analyzovali požadavky zadání nám přiděleného projektu. Navrhli jsme si jak bude výsledná databáze vypadat. Následně jsme si rozdělili mezi sebou úkoly, na kterých bude každý z nás pracovat.




Obrázek 2: Relační model databáze pro inzertní aplikaci

Advertisement obsahuje data o jednotlivých inzerátech. Category obsahuje název kategorie a slouží ke kategorizaci inzerátu pro větší přehlednost. Picture reprezentuje obrázky uložené v databázi, obsahuje jediný atribut - image, který má v sobě uloženou adresní cestu k obrázku na serveru. Reply slouží k uložení zpráv mezi uživateli v aplikaci. Conversation obsahuje jedinečný název pro konverzaci mezi uživateli o určitém inzerátu a všechny zprávy, které si uživatelé o tomto inzerátu napíší. User obsahuje přihlašovací údaje o všech uživatelích v aplikaci. Contact

obsahuje kontaktní údaje na uživatele.

### 6.2.2 Vytvoření modelů Category a Advertisement

Nejprve jsem vytvořil modely Kategorie a Inzerát, ve kterých jsem napsal validace vstupu pro jednotlivé atributy.

```
class Category < ActiveRecord::Base
  has_many :advertisements, dependent: :destroy

  validates :name, presence: true, length: {minimum: 1, maximum: 200}
end
```

Výpis 7: Validace modelu Category

Validace modelu Category začíná definováním vazby mezi modely Advertisement a Category. Jedna instance modelu Category může mít více instancí typu Advertisement. Příkazy **dependant: :destroy** slouží k tomu, když bychom smazali záznam z databáze, smažou se také všechny záznamy typu Advertisement, které byly propojené se smazaným záznamem Category.

Pokračuji s validací atributu name. Atribut name musí být vyplněn a jeho délka se musí pohybovat mezi 1 až 200 znaky.

```
class Advertisement < ActiveRecord::Base
  belongs_to :category

  validates :description, presence: true, length: {maximum: 200, minimum:
      1}
  validates :price, presence: true, numericality: {only_integer: true}
  validates :post_date, :end_date, :user_id, presence: true

  validate :deadline_is_possible
  validate :validate_price

  before_validation :make_user
  after_create :set_sold_initial_status

  def deadline_is_possible
    return if [post_date.blank?, end_date.blank?].any?
    if post_date > end_date
      errors.add(:end_date, I18n.t('AttentionDate'))
```

```
      end
    end

    def set_sold_initial_status
      sold = false
    end

    def validate_price
      if !price.is_a? Integer
        errors.add(:price, I18n.t('AttentionPrice'))
      end
    end
  end
```

Výpis 8: Validace modelu Advertisement

Nejprve jsem zapsal vztah mezi Advertisement a Category. Každá jedna instance typu Advertisement patří k jedné instanci typu Category.

Atribut description slouží k popisům inzerátů. Musí být vyplněn a jeho délka se musí pohybovat v rozmezí 1 až 200 znaků. Atribut price bude obsahovat celočíselnou cenu zboží nabízeného v inzerátu. Příkaz **numericality: {only_integer: true}** znamená, že povolíme pouze data typu integer. U atributů post_date, end_date a user_id musí být pouze vyplněny, žádné jiné omezení nemají.

Pokračujeme validací, jestli má vůbec smysl vytvářet instanci typu Advertisement. Toto si ověříme pomocí metody **deadline_is_possible**, která slouží k ověření, zda je datum zveřejnění inzerátu(post_date) mladšího data a času než jeho konec platnosti(end_date). Pokud bude zveřejnění inzerátu později než je jeho konec platnosti, vyhodí chybu. Jinak vše pokračuje dále.

Metoda **validate_price** slouží k ověření, zda uživatel vyplnil cenu za zboží jako datový typ integer. Pokud nevyplnil jako integer vyhodí chybu. Jinak pokračuje dále.

Metoda **set_sold_initial_status** slouží k nastavení hodnoty atributu sold na false, tedy na status neprodaný.

Metoda **make_user** slouží k vytvoření nového uživatele, pokud chce neregistrovaný uživatel vytvořit nový inzerát. Tomuto bych se chtěl věnovat později.

### 6.2.3 Přidání modelu Picture

Přidal jsem další model Picture, který obsahoval jednotlivé informace o jednom obrázku uloženém na serveru. Každý obrázek může být přiřazen k jednomu inzerátu, proto obsahuje cizí klíč inzerátu, ke kterému je přiřazen. Obsahuje také informaci o datu a času vytvoření a úpravy. Poslední atribut, který obsahuje je název souboru na serveru. Název obrázku je automaticky generován. K nahrání souboru na server jsem využil gemu Carrierwave. V Carrierwave jsem

nastavil zmenšení obrázku tak, aby šířka obrázku byla maximálně 250 pixelů a výška pokud možno 290 pixelů.

```
...
version :thumb do
   process :resize_to_fit => [290, 250]
end
...
```

Výpis 9: Zmenšení obrázku třída ImageUploader

Obrázky budou ukládány do složky uploads/název_třídy/id/id.

```
...
def store_dir
   "uploads/#{model.class.to_s.underscore}/#{mounted_as}/#{model.id}
end
...
```

Výpis 10: Carrierwave složka pro ukládání obrázků

### 6.2.4 Stránkování

Vytvořil jsem stránkování ve views, kde jsou vypsány seznamy inzerátů přes gem Kaminari. Jak jsem stránkování provedl je možno vidět ve Výpise č.X. Views, kde jsou vypsány seznamy inzeráty jsou u zobrazení všech inzerátů nezávisle na jejich kategorii (domovská stránka) a při zobrazení detailu kategorie, ve kterém jsou vypsány všechny inzeráty spadající do vybrané kategorie.

### 6.2.5 Model Contact

Provedl jsem test modelu Contact a zvalidoval jeho vstupy. A následně jsem vytvořil Complex form na editaci uživatele, který měnil údaje u uživatele i v jeho kontaktních údajích najednou, pokud uživatel zadá správné heslo.

```
class Contact < ActiveRecord::Base
   has_one :user

   validates :firstname, presence: true, length: {minimum: 2, maximum: 50}
   validates :surname, presence: true, length: {minimum: 2, maximum: 50}
   validates :phonenumber, :format => {:with => /\A[+]?[0-9\-\ ]{3,15}\z/},
      :allow_blank => true
```

```
    validates :city, presence: true, length: {minimum: 2, maximum: 50}
    validates :street, presence: true, length: {minimum: 2, maximum: 50}
    validates :zipcode, presence: true, numericality: {only_integer: true}

  end
```

Výpis 11: Validace modelu Contact

Ve validaci modelu Contact jsem nejprve určil vazbu s tabulkou User. Jedna instance typu Contact má vazbu na jednu instanci typu User.

U atributů firstname, surname, city a street zajišťuji, že u každého objektu typu Contact budou tyto údaje vždy vyplněny. A délka těchto stringů musí být od 2 do 50 znaků.

Atribut phonenumber bude obsahovat telefonní číslo. Telefonní číslo v kontaktu uživatele není povinné, tuto vlastnost jsem zajistil příkazem **:allow_blank => true**.

Poslední atribut zipcode bude obsahovat směrovací číslo adresy uživatele. Musí být vyplněno. Vždy bude obsahovat pouze celočíselné data, abych toho dosáhl využil jsem příkazu **numericality: => {only_integer: true}**.

### 6.2.6 Lokalizace

Webovou aplikaci lze přeložit do několika jazyku. K přeložení do jednoho jazyka je potřebný jeden .yml soubor. Yml soubor obsahuje dvojce záznamů klíč: hodnota. Lokalizační soubory jsou uloženy v config/locales/zkratka jazyka/zkratka jazyka.yml.

```
...
Sold: "Prodano"
NotSold: "Neprodano"
Sell: "Prodat"
Return: "Vratit"
...
```

Výpis 12: Ukázka z českého lokalizačního souboru

V ukázce kodu výše lze vidět záznamy klíč: hodnota v .yml souboru. Abych přeložil například odkaz na prodání inzerátu (klíč Sell) a tím ho stáhl z inzerce je nutné:

```
...
<% if advertisement.sold == false %>
  <%= link_to t('Sell'), sell_category_advertisement_path(advertisement.
      category_id, advertisement) %>
<% else %>
  <%= link_to t('Return'), sell_category_advertisement_path(advertisement.
      category_id, advertisement) %>
```

```
<% end %>
...
```

Výpis 13: Prodání inzerátu a jeho následné možné vrácení do inzerce s překladem

V ukázce výše můžeme vidět způsob, jakým lze prodat inzerát (tím ho stáhnout z prodeje) a zase ho vrátit (tím ho začít opět inzerovat).

Pokud má atribut sold instance Advertisement nastavenu hodnotu false, znamená to, že není prodaný a v tom případě ho můžeme prodat. Proto jsem přidal **link_to**. Tento příkaz přidá do view odkaz na prodání inzerátu. Aby byl tento odkaz přeložen i do češtiny popřípadě jiných jazyků, musel jsem přidat příkaz t('Sell'), který slouží překladu podle klíčového slova z českého lokalizačního souboru Sell. V našem případě místo t('Sell') bude string s hodnotou 'Prodat'.

Pokud atribut sold instance Advertisement bude mít nastavenou hodnotu na true. Bude možnost pouze inzerát vrátit, protože již prodaný byl. Proto jsem přidal odkaz na vrácení inzerátu s překladem t('Return'), který jako string vrátí hodnotu 'Vrátit'.

### 6.2.7 Vytvoření inzerátu neregistrovaným uživatelem

Přidal jsem funkci, která umožňuje neregistrovanému uživateli napsat inzerát, ale s jeho současným zaregistrováním. Neregistrovaný uživatel stejně jako registrovaný uživatel přejde na detail inzerátu. V detailu inzerátu bude mít možnost stejně jako registrovaný uživatel reagovat na inzerát. Jakmile bude chtít reagovat na inzerát objeví se mu formulář, ve kterém může napsat zprávu uživateli, který inzerát inzeruje. Jediný rozdíl, který oproti registrovanému uživateli je ten, že musí navíc vyplnit svůj email.

```
class Advertisement < ActiveRecord::Base
   ...
   before_validation :make_user
   ...
   def make_user
      if User.find_by(:email => self.user_email) == nil
         self.user_id = User.create(:email => self.user_email, :password => '
             123456789').id if self.user_email.present? && self.post_date <
             self.end_date
      end
   end
   ...
end
```

Výpis 14: Vytvoření uživatele při vytvoření inzerátu neregistrovaným uživatelem

Funguje tak, že před validací nové instance typu Advertisement se zavolá metoda **make_user**, která má za úkol vytvořit nového uživatele. Nejprve v inzertní aplikaci vycházíme z toho, že email uživatele musí být unikátní v rámci celého systému. Proto nejprve ověřím, zda se již nenachází uživatel s emailem, který vložil neregistrovaný uživatel při vytvoření inzerátu. Pokud email nebyl nalezen, musí být ještě splněna podmínka, že email byl vůbec vyplněn a zveřejněný datum inzerátu byl před datem konce platnosti inzerátu. Jakmile všechny tyto podmínky jsou splněny, můžeme vytvořit účet s provizorním heslem, které by měl uživatel změnit po přihlášením do systému.

### 6.2.8 Komunikace založena na zprávách mezi uživateli

Komunikace zpráv založena na zprávách v systému ukládá všechny zprávy, které si mezi sebou posílají dva uživatelé o jednom inzerátu.

Nejprve jsem vytvořil model v databázi a zvalidoval ho. V controlleru od Replies (jedná se o objekt, který v sobě udržuje informace o každé poslané zprávě v rámci systému) nejprve vytvořím unikátní název konverzace.

```
class Reply < ApplicationController
   def create
      ...
      conversation_name = @author.email + ' ' + @asker.email + ' ' +
         @advertisement.description.truncate(20) + '...'
      ...
   end
end
```

Výpis 15: Vytvoření názvu konverzace při vytvoření nové zprávy

Při vytvoření nové zprávy se vytvoří unikátní název konverzace z emailu autora inzerátu, emailu uživatele, který reaguje na inzerát a prvních dvaceti znaků popisujících inzerát. Dále musíme ověřit, zda se má zpráva vložit do již existující konverzace nebo do nové konverzace.

```
class Reply < ApplicationController
   def create
      ...
      conver = Conversation.find_by(:name => conversation_name)
      if conver.nil?
         ...
      end
      ...
   end
end
```

Výpis 16: Ověření existence konverzace v systému

Pokusíme se najít v databázi konverzaci s unikátním názvem, který jsme dříve vytvořili. Pokud konverzace existuje s tímto unikátním názvem existuje, tak do ní vložíme novou zprávu. Pokud konverzace neexistuje, vytvoříme novou konverzaci, do které vložíme nově vzniklou zprávu. Uživatele mohou své zprávy vidět v konverzacích. Po rozkliknutí názvu konverzace uvidí výpis všech zpráv s časem odeslání, odesilatelem a obsahem zprávy.

### 6.2.9 Dynamické přidávání fotek

V aplikaci má jít u každého uživatele přidat až deset obrázků. Nicméně původně se vytvořili ke každé instanci typu Advertisement vytvořeno deset instancí typu Picture. Pokud chtěl uživatel přidat další fotku, přidal k do již vytvořeného objektu nahrál soubor. Kvůli úspoře místa a nevyužitosti záznamů v databázi jsem došel k závěru, že lepší bude toto změnit a přidávat až deset fotek dynamicky.

U detailu inzerátu uživatel, který je autorem inzerátem může přidat obrázek. Tímto se vytvoří nová instance typu Picture, do které nahraje obrázek. Přidal jsem do typu Advertisement kolekci, která bude obsahovat všechny fotky pro jeden inzerát.

```
class Picture < ApplicationController
   ...
   def create
      @picture = Picture.new(picture_params)
      @category = Category.find(params[:category_id])
      @advertisement = Advertisement.find(params[:advertisement_id])
      if @advertisement.pictures.size <10
         @user = current_user
         if @picture.save
         flash[:notice] = t('InfoPicture')
         @advertisement.pictures<<@picture
         end
      else
         flash[:error] = t('ErrorPicture')
      end
      respond_with(@category, @advertisement)
   end
   ...
end
```

Výpis 17: Úprava na dynamické přídávání až deseti fotek

Všechnu výše zmíněnou problematikou jsem vyřešil v metodě create controlleru Picture. Nejprve jsem si do třídní proměnné picture vytvořil objekt typu Picture z informací získaných z view formuláře. Dále jsem si vytvořil objekty category (typu Category) a advertisement (typu Advertisement). Data jsem si do těchto dvou objektů získal z databáze. Abych našel potřebné záznamy v databázi využil jsem dat odeslaných z formuláře ve view. Následně zjišťuji zda kolekce v objektu advertisement obsahuje již deset objektů typu Picture.

Pokud neobsahuje tak, se uloží obrázek do databáze a přidáme tento objekt do kolekce k advertisement.

Pokud obsahuje již obsahuje deset obrázků vypíše se chybu.

Na konec přesměruji na detail inzerátu.

Opravení a zlepšení funkce přidávání až deseti fotek a jejich následného zmenšení.

### 6.2.10 Odpověď na inzerát

Každý uživatel může odpovědět na inzerát. Ať už je registrovaný nebo ne. Pokud neregistrovaný uživatel bude chtít odpovědět na inzerát, tak ho systém také registruje.

```
%p.btn.btn-default{:type => "button"}= link_to t('HaveInterested') ,
    new_category_advertisement_reply_path(@category,@advertisement)
```

Výpis 18: Část view obsahující přesměrování na napsání odpovědi na inzerát

Ukázka výše slouží k tomu, že v do paragrafu p v HTML vloží tlačítko, které bude obsahovat odkaz na přesměrování na stránku, kde může uživatel napsat odpověď na inzerát. K přesměrování potřebujeme znát třídní proměnnou category a advertisement, které obsahují informace o kategorii, ve které se nachází žádaný inzerát a informace o inzerátu samotném.

### 6.2.11 Přidání stavu inzerátu

Přidání stavu inzerátu na prodáno nebo vráceno. Uživatel může určit, kdy je inzerát již prodaný a tím ho stáhnout z inzerce.

Stav inzerátu jsem přidal přes migraci. Přes migraci jsem přidal do databáze nový atribut, který bude obsahovat potřebný stav. Přes příkaz v příkazové řádce jsem přidal migraci do programu.

```
rails generate migration AddSoldToAdvertisement sold:boolean
```

Výpis 19: Příkaz do příkazové řádky pro vytvoření migrace

Výše lze vidět příkaz, který je nutno napsat pro vytvoření migrace v aplikaci. Po stisknutí enteru, se vygeneruje migrace, pokud je ovšem vše v pořádku. Pokud v pořádku něco není buď se vyhodí chyba nebo se vytvoří pouze třída s prázdnou metodou change, kde je nutno ručně dopsat co se přesně má provést. Je možnost takovéto migrace dělat vícero operací než jen jednu.

```
class AddSoldToAdvertisements < ActiveRecord::Migration
  def change
    add_column :advertisements, :sold, :boolean
  end
end
```

Výpis 20: Obsah vytvořené migrace v aplikaci

Potom co jsem měl migraci ve tvaru, který je výše, jsem napsal do příkazové řádky příkaz pro změnu databáze dle vytvořené migrace. Tento příkaz tedy přidá do databáze atribut sold, který bude datového typu boolean.

```
rake db:migrate
```

Výpis 21: Příkaz pro změnu databáze pomocí migrace

#### 6.2.12 Otestování a validace aplikace

Finální napsaní testů pro aplikaci. Validace všech vstupu. A celkové otestování funkčnosti aplikace.

### 6.3 Zhodnocení práce

Na projektu inzertní aplikace jsme pracovali od úplného začátku. Navrhli jsme databázi, naprogramovali hlavní funkčnosti aplikace. Konzultant z firmy nad naší prací dohlížel a radil nám jak náš kód zlepšit. Tento projekt jsme dopracovali do fáze, kdy bylo nutné ještě přidat šablonu pro změnu stylování stránky nebo ještě popřípadě provést mírné úpravy po dohodě s klientem, pro kterého byla tato aplikace vyvíjena.

Mezi největší problémy, se kterými jsem se setkal při vývoji této aplikace, byly problémy s gemem pro nahrávání souborů a jeho celkovým zprovozněním a vytvoření dynamického přidání až desíti fotek k jednomu inzerátu. Dále také komunikační systém mezi uživateli založený na zasílání zpráv mezi sebou umístěných v konverzacích.

# 7 Webová aplikace bankovních importů

Mým posledním projektem byla webová aplikace bankovních importů. Byl jsem přidělen do týmu o velikosti třech stážistů. Na naší práci dohlížel konzultant z firmy.

## 7.1 Zadání práce

Tato webová aplikace má sloužit k přehledům a statistikám o bankovních převodech k bankovním účtům uživatelů. Má pomoci především firmám nebo lidem s několika účty k většímu přehledu nad finančními pohyby z a na své účty.

Předtím, než uživatel začne používat tuto aplikaci je nutné, aby se nejprve zaregistroval. Následně vloží do aplikace informace o účtech jaké chce sledovat. Mezi informacemi nezbytně nutnými pro úspěšné fungovaní této aplikace je, aby uvedl email na, který mu chodí bankovní výpisy od své banky a číslo bankovních účtů, které chce nechat sledovat v této aplikaci. Další důležitou věcí je pro uživatele, aby přesměroval výpisy od svých bank emailovou schránku patřící pod webovou aplikaci. Emailové zprávy, které budou přicházet od uživatele (jeho emailu), budou automaticky parsovány a důležitá data budou ukládány do databáze jako transakce pod bankovní účet, ke kterému náleží.

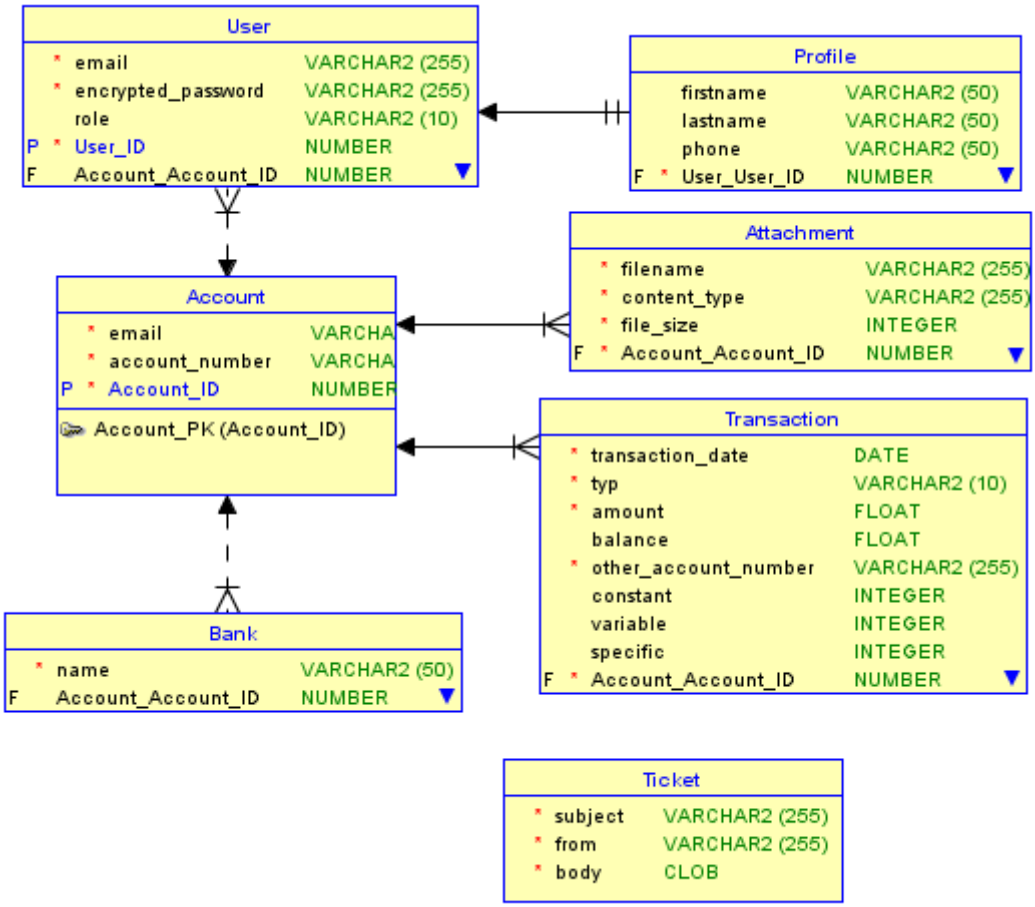
K údajům o bankovních výpisech bude mít přístup pouze uživatel, kterému daný účet patří. Vyjma uživatele role administrátor, který při vzniklém problému, může za uživatele nastavit jeho uživatelský účet.

## 7.2 Moje práce

V této části se budu zabývat výhradně částmi, které jsem v projektu webové aplikace bankovní importy zpracoval já. Dovolil jsem si vynechat některé části, na kterých jsem pracoval, z důvodu podobnosti s již řešenými problémy z inzertní aplikace v této bakalářské práci uvedené.

### 7.2.1 Analýza zadání projektu

Nejdříve jsme s mými kolegy analyzovali požadavky zadání nám přiděleného projektu. Navrhli jsme si jak bude výsledná databáze vypadat. Následně jsme si rozdělili mezi sebou úkoly, na kterých bude každý z nás pracovat.




Obrázek 3: Relační model databáze pro aplikaci bankovní importy

Tabulka Account obsahuje email a číslo účtu. Email slouží jako adresa, z které budou přicházet zprávy o bankovních výpisech na email aplikace bankovních importů. Číslo účtu je zde uchováváno z důvodu, kdy by měl uživatel více bankovních účtů na jednom emailu. Aby se dalo jednoznačně rozlišit ke kterému bankovnímu účtu výpisy přiřadit.

Tabulka Bank obsahuje pouze název banky, slouží ke kategorizaci jednotlivých účtu dle bank. Druh banky zaznamenáváme z důvodu, že každá banka zasílá emaily v jiné podobě, tudíž jiný tvar parse metod. Jedná se tabulkový číselník.

Tabulka User obsahuje přihlašovací údaje uživatelského účtu včetně role jako bude uživatelský účet mít.

Tabulka Profile obsahuje kontaktní údaje uživatele.

Tabulka Attachment obsahuje všechny data o přílohách, která byla zaslána spolu s emailem od uživatele.

Tabulka Transaction obsahuje data o jednotlivých transakcích, která přišla ve výpisech od uživatele. Jako například kdy byla transakce provedena, částku, protiúčet, atd.

Tabulka Ticket obsahuje jednotlivé data z emailu. Jako například od koho byl email původně odeslán, předmět emailové zprávy a tělo zprávy.

### 7.2.2 Autorizační systém

Při vytvoření autorizačního a a jsem využil gemu Cancan. Po nainstalování gemu Cancan jsem vložil do tabulky user atribut role, do kterého jsem určoval jaké role uživatel je.

Nastavil jsem v třídě Ability role User (běžně registrovaný uživatel) a Admin (administrátor stránky).

```
class Ability
  include CanCan::Ability

  def initialize(user)
   user ||= User.new
   can :create, User
   if user.role == 'user'
       can :create, Account
       can :manage, Account, user_id: user.id
       can :manage, User, id: user.id
       can :manage, Profile, user_id: user.id
       can :read, Transaction
       can :manage, Attachment, account: {user_id: user.id}
   end
   can :manage, :all if user.role == "admin"
  end
   end
```

Výpis 22: Nastavení základních pravomocí jednotlivých rolí v třídě Ability

Ukázka výše slouží k přidání prvku autorizace do aplikace.

Nejprve nastavíme, aby se neregistrovaný uživatel mohl pouze registrovat (příkazem **can :create, User**). Kvůli restrikci přístupu do pro něj nepřístupných částí aplikace.

Pokud je uživatel role user, může vytvářet nové bankovní účty ke svému uživatelskému účtu. Může spravovat svoje přihlašovací údaje, měnit své kontaktní údaje, číst své transakce u svých bankovních účtů a má přístup do přiložených souborů, které mu od jeho bank přišly na email a tím i do aplikace bankovních importů.

Pokud je uživatel role admin, může spravovat všechny data. Může spravovat jak svůj účet a data k němu přiřazená (stejně jako normální uživatel). Narozdíl od normálního uživatele může navíc spravovat data i ostatních uživatelů.

Pokud se registruje nový uživatel, implicitně se k němu přiřadí role user.

```
class User < ActiveRecords::Base
   ...
   after_creation :set_default_role

   def set_default_role
      if self.role.blank?
         self.role = 'user'
      end
   end
   ...
end
```

Výpis 23: Při vytvoření uživatele implicitní role user

Pokud chceme vytvořit nového uživatele role Admin musí se to provést přes příkazovou řádku nebo ručním přidáním do databáze. Následně jsem upravil views, aby každá role měla přístup pouze k funkcím, na které má právo.

```
...
if current_user.role == 'admin'
   %li#fat-menu.dropdown
     %a.dropdown-toggle{:data => "", :href => "#", :toggle => "dropdown"}
       = t('adminPage')
       %b.caret
     %ul.dropdown-menu
       %li= link_to t('allUsersOverview'), users_path
       %li= link_to t('Banks'), banks_path
...
```

Výpis 24: Omezení pravomocí v aplikaci pomocí roli ve views

Výše lze vidět, že když je uživatel přihlášen do role admin, tak uvidí odkazy na správu uživatelů a bank.

### 7.2.3 Vytvoření profilu a jeho propojení s uživatelem

Nejprve jsem měl vytvořenou třídu User, reprezentující všechny uživatele. Poté jsem vytvořil třídu Profile, která bude obsahovat kontaktní údaje na uživatele. Jak lze podle návrhu databáze vidět, každý objekt typu User má k sobě vždy přiřazenu jednu instanci typu Profile. Abych

nastavil, že po vytvoření nového uživatele, bude tento nově vzniklý objekt obsahovat vazbu na objekt typu Profile. Napsal jsem následující kód do modelu třídy User:

```
class User < ActiveRecord::Base
   ...
   after_create :make_a_profile
   ...
   def make_a_profile
       profile_instance = Profile.new
       profile_instance.save!
       self.profile_id = profile_instance.id
       self.save!
   end
   ...
end
```

Výpis 25: Vytvoření instance typu Profile po vytvoření instance typu User

Z výpisu výše vyplývá, že po vytvoření instance typu User se spustí metoda **make_a_profile**. V této metodě se vytvoří nový objekt typu Profile, který se uloží do databáze. Jeho ID se vloží do objektu User, z kterého byla metoda **make_a_user** zavolána, poté se objekt typu User uloží a tím je přiřazen objekt typu Profile k objektu typu User.

### 7.2.4 Povýšení uživatele na admina

Aby mohl být vytvořen další uživatel, jehož role bude administrátor, musí se nejprve zaregistrovat jako běžný uživatel. Tím bude mít nově vzniklý uživatel roli user, tedy běžného uživatele.

Nejprve jsem vytvořil přehled všech uživatelů, kde má přístup pouze uživatel role admin. U každého uživatele uvidí odkaz na jeho povýšení z role user na roli admin nebo opačně při nutnosti degradace jeho autorizace v rámci aplikace.

Nejprve jsem musel zajistit, aby nemohl administrátor změnit svou roli z admin na user, pokud bude jediným administrátorem v rámci aplikace.

```
class UsersController < ApplicationController
   ...
   def index
      @usersCollection = User.all.where(:role => 'admin')
   end
   ...
end
```

Výpis 26: Počet všech uživatelů role admin v controleru

Kolekce usersCollection bude obsahovat všechny uživatele, kteří jsou role admin.

```
...
- for user in @users
  %tr
    %td= user.email
    %td= user.role
    -profile = Profile.find_by(:id => user.profile_id)
    %td= profile.firstname
    %td= profile.lastname
    %td= profile.phone

    -if user.role == 'admin'
      -if @usersCollection.size > 1
        %td= link_to 'Demote', promote_user_path(user.id)
      -else
        %td
    -elsif user.role == 'user'
      %td= link_to 'Promote', promote_user_path(user.id)
...
```

Výpis 27: Výpis všech uživatelů v systému včetně možnosti změnit roli ve views

Vypíší se zde všichni uživatelé. U každého uživatele se vypíše jeho email, role, křestní jméno, příjmení a telefon. Pokud bude uživatel role admin, a počet uživatelů role admin bude větší než 1, může tohoto uživatele degradovat na roli user. Pokud bude uživatel role user, může tohoto uživatele povýšit na roli admin.

### 7.2.5 Parse dat z emailu

Data, se kterými tato aplikace pracovala byly získávány ze zaslaných emailů od uživatelů. Proto bylo nutné vyřešit, jak tyto data z emailů zparsovat do přijatelné podoby. Každá banka má jiné formáty emailů obsahující bankovní výpisy za poslední měsíc. Některé banky zasílají bankovní výpisy v těle emailu a další zase v přiloženém souboru emailu (buďto .txt nebo .pdf). Proto bylo nutné pro každou banku napsat parsování, jak data z těchto emailu zpracovat. Já jsem zpracoval parsování na bankovní výpisy banky Raiffeisenbank a Fio banka.

### 7.2.6 Export dat do XML souboru

Uživatel by měl mít možnost stáhnout si výpisy o svých transakcích. Buď výpis všech transakcí během měsíce, výpis všech transakcí během celé doby nebo výpis všech transakcí mezi dvěma

daty. Princip funkčnosti je v podstatě u všech exportů do XML souboru stejný, proto vyberu ten nejtěžší a nejzajímavější a to výpis mezi dvěma daty.

Nejprve jsem přidal do view zobrazující detail bankovního účtu:

```
...
= search_form_for(@search, format: :xml) do |f|

  = f.label :transaction_date_lteq
  = f.date_select :transaction_date_lteq

  =f.label :transaction_date_gteq
  = f.date_select :transaction_date_gteq
  = hidden_field_tag(:account, @acc.id)
  = f.submit 'Vytvorit XML'
...
```

Výpis 28: Formulář pro vytvoření XML mezi dvěma datumy

Ve výpisu výše je úryvek kódu v Ransack gemu, který vygeneruje formulář na výběr počátečního a konečného data, mezi kterými bude uživatel chtít vygenerovat XML soubor. Dále je tam přidán skrytý parametr, který se odesílá po odeslání formuláře a bude posílat ID bankovního účtu, který budu potřebovat později.

Po odeslání formuláře se parametry odešlou do Transaction controlleru do metody index, kvůli tomu, že gem Ransack automaticky přesměrovává na výpis všech transakcí, pokud hledáme mezi transakcemi, proto bylo nutné upravit tuto metodu, kde jsem napsal metodu pro export transakcí do XML a následně kód pro stažení výsledného XML souboru.

```
class TransactionsController < ApplicationController
   respond_to :html, :xml
   ...
   def index
       ldate_day = params[:q]['transaction_date_lteq(3i)']
       ldate_month = params[:q]['transaction_date_lteq(2i)']
       ldate_year = params[:q]['transaction_date_lteq(1i)']
       gdate_day = params[:q]['transaction_date_gteq(3i)']
       gdate_month = params[:q]['transaction_date_gteq(2i)']
       gdate_year = params[:q]['transaction_date_gteq(1i)']

       account = Account.find_by(:id => params[:account])

       ldate = Date.new(ldate_year.to_i, ldate_month.to_i, ldate_day.to_i)
       gdate = Date.new(gdate_year.to_i, gdate_month.to_i, gdate_day.to_i)
```

```
        @transactions = Transaction.all.where('transaction_date < ? AND
            transaction_date > ? AND account_id = ?', gdate, ldate, account.id
            )
        respond_to do |format|
          format.html
          format.xml { send_data @transactions.to_xml,
              :type => 'text/xml; charset=UTF-8;',
              :disposition => "attachment; filename=transactions.xml"
             }
        end
    end
    ...
end
```

Výpis 29: Export do XML mezi dvěma datumy v metodě index v controlleru Transaction

V ukázce výše nejprve určím, že tento controller může vykreslovat jak HTML tak i XML soubory pomocí příkazu **respond_to :html, :xml**.

V metodě index načtu informace o datech rozložených na jeho jednotlivé části (den, měsíc, rok) z parametrů poslaných z formuláře. Uložím tyto parametry do string proměnných.

Skrytý parametr, který jsem předtím vložil do formuláře obsahuje ID bankovního účtu, kterého se týká právě tento export do XML souboru. Toto ID využiji k tomu, abych nalezl data z databáze o tomto účtu a uložil je do lokální proměnné account.

Vytvořím dvě proměnné typu Date - ldate a gdate, které poskládám z proměnných, které obsahují string hodnoty obsahující dny, měsíce a roky datumů. Při vytváření objektu typu Date, musím nicméně do parametrů od Date vložit hodnoty typu integer a ne string proto využiji metody **to_i**, která mi hodnotu string konvertuje na integer.

Následně vytvořím kolekci transactions, která bude obsahovat všechny transakce, jejichž datum je mezi dvěma daty, které zadal uživatel a patří bankovnímu účtu, nad kterým uživatel poslal požadavek na export do XML souboru.

Nakonec zvolím formáty, které může uživatel nad touto metodou zavolat. Pokud bude požadavek na HTML, vykreslí se view jako obvykle. Pokud bude chtít uživatel XML soubor, pomocí příkazu **send_data** se stáhnou data transactions konvertované do XML pomocí metody **to_xml**. Nicméně je ještě nutné specifikovat další informace, jinak dojde při konverzi k chybě. Důležitým je kódování řetězců UTF-8, bez kterého není možné správně překonvertovat do XML souboru. Pokud bychom chtěli aby se soubor pouze zobrazil a nestahoval je možné místo **send_data** použít metody **render**.

### 7.2.7 Nahrávání souboru k bankovním účtům

Uživatel by měl mít právo nahrát si ke každému svému bankovnímu účtu soubor s příponou .txt nebo .pdf, který bude obsahovat informace o bankovních transakcích. Na tomto úkolu jsem pracoval s dalším stážistou. Můj kolega provedl všechna potřebná nastavení, aby bylo možné soubory nahrávat a určil s jakou příponou lze soubory nahrát. Mým úkolem bylo naprogramování přístupu k přílohám, následně jejich stažení a zobrazení.

V Attachment controlleru v metodě index načtu do kolekce attachments všechny přiložené soubory, které patří pod bankovní účet, u kterého chceme prohlížet přiložené soubory. Poté v Attachment index view:

```
...
- @attachments.each do |attachment|
%tr
   %td= attachment.fileUp.file.identifier
   %td= link_to t('.Preview'), '/attachment/fileUp/'+attachment.id.to_s+'/'+
       attachment.fileUp.file.identifier
   %td= link_to t('Download'), download_user_account_attachment_path(@user,
       @account, attachment)
   %td= link_to t('Destroy'), [@user,@account,attachment], method: :delete,
       data: { confirm: t('AreYouSure') }
...
```

Výpis 30: Výpis všech přiložených souborů u bankovního účtu ve view index

U každého souboru se vypíše název souboru, odkaz na jeho zobrazení (aby bylo možné soubor zobrazit je nutné specifikovat jeho adresu na disku, která se skládá z úvodních složek, ID přílohy a názvu souboru), odkaz na jeho stažení a metoda na jeho smazání. Aby bylo možné stáhnout soubor, využívá se odkazu na metodu download, která je v controlleru Attachment.

```
class AttachmentsController < ApplicationController
...
   def download
      attachment = Attachment.find_by(:id => params['id'])
      send_file('public/attachment/fileUp/'+attachment.id.to_s+'/'+attachment
          .fileUp.file.identifier)
   end
...
end
```

Výpis 31: Metoda download v controlleru Attachment umožňující stáhnutí přílohy

V ukázce výše lze vidět metoda download, ve které se nejprve načtou data o příloze, kterou budeme stahovat. Poté metoda send_file umožní uživateli stažení přílohy.

## 7.3 Zhodnocení práce

Na projektu bankovních importů jsme pracovali taktéž od úplného začátku. Navrhli jsme databázi, naprogramovali hlavní funkčnosti aplikace. Konzultant z firmy nad naší prací dohlížel a také s námi na tomto projektu spolupracoval. Tento projekt jsme dopracovali do fáze, kdy byla aplikace z velké části funkční, bylo ještě nicméně nutné dodělat další parsy pro více bank, přidat šablonu stylování stránky a dále provést další úpravy po dohodě s klientem, pro kterého byla tato aplikace vyvíjena.

Mezi největší problémy, které jsem se při vývoji této aplikace setkal, patřilo bezpochyby nahrávání souborů k bankovním účtům, jejich následné stáhnutí a zobrazení nebo naprogramování stáhnutí nově vytvořeného XML souboru, kde nastal problém se špatným kódováním souboru.

# 8 Závěr

Znalosti, které jsem se naučil při mém studiu na Vysoké škole Báňské v Ostravě a využil je při mé bakalářské praxi byly například z předmětů Vývoj informačních systémů (VIS) a Databázové a informační systémy (DAIS), ze kterých jsem využil postupy při analýze a návrhu informačních systémů. Úvod do systémového inženýrství (SWI), z kterého jsem využil metodiky vývoje softwarového díla. Dále předmět Skriptovací programovací jazyky a jejich aplikace (SPJA), která mě seznámila se základy skriptovacích jazyků do nichž patří i jazyk Ruby.

Znalosti, které jsem získal během mé bakalářské praxe jsou osvojení si webového frameworku Ruby on Rails, práce s verzovacím systémem GIT, vyzkoušení si programování na větších projektech v týmu, práce s operačním systémem Linux.

Tuto bakalářskou praxi hodnotím jako velmi povedenou. Přinesla mi spoustu životních i pracovních zkušeností, které mohu dále využít při mém dalším uplatnění na trhu práce.

Během své bakalářské praxe jsem přispěl firmě svou prací na dvou větších projektech. Jednalo se o webovou Inzertní aplikaci a webovou aplikaci Bankovních importů.

Miroslav Babral

# Literatura


[1] Odbor evropských fondů hl. města Prah. In: *Evropské fondy Praha* [online]. Praha, 2009 [cit. 2016-02-29]. Dostupné z: http://www.prahafondy.eu/cz/oppa/pro-prijemce/325_pomucka-pro-urceni-velikosti-podniku.html

[2] Webová stránka webové aplikace sMoneybox. In: *JVK Consulting s.r.o.* [online]. [cit. 2016-03-16]. Dostupné z: http://www.smoneybox.com/

[3] Webová stránka sÚčto In: *JVK Consulting s.r.o.* [online]. [cit. 2016-03-16]. Dostupné z: https://www.sucto.cz/

[4] Webová stránka informačního systému Hedurio In: *Railsformers s.r.o. Tours* [online]. [cit. 2016-03-16]. Dostupné z: https://hedurio.com/

[5] Webová stránka firmy Rainbox tours In: *Rainbow Tours* [online]. [cit. 2016-03-16]. Dostupné z: http://rainbowtours.cz/

[6] Firma Railsformers s.r.o. In: *Railsformers s.r.o.* [online]. Ostrava [cit. 2016-02-29]. Dostupné z: http://railsformers.com

[7] Oficiální stránka jazyka Ruby. In: *Ruby* [online]. [cit. 2016-02-29]. Dostupné z: https://ruby-lang.org/en/about/

[8] Oficiální stránka Ruby on Rails. In: *Ruby on Rails* [online]. [cit. 2016-02-29]. Dostupné z: http://rubyonrails.org/

[9] Github kanál zabývající se řešením problému nebo chyb v RoR. In: *GitHub* [online]. [cit. 2016-02-29]. Dostupné z: https://github.com/rails/rails/issues

[10] Průzkum zabývající se využitím webových frameworků. In: *BuiltWith* [online]. Sydney(Austrálie): BuiltWith® Pty Ltd, 2016 [cit. 2016-02-29]. Dostupné z: http://trends.builtwith.com/framework

[11] Webová stránka obsahující všechny gemy Ruby on Rails. In: *RubyGems* [online]. [cit. 2016-02-29]. Dostupné z: https://rubygems.org/pages/about

[12] Informace o OS Ubuntu. In: *Ubuntu* [online]. London(Velká Británie): Canonical Group Limited [cit. 2016-02-29]. Dostupné z: http://www.ubuntu.com/about/about-ubuntu

[13] Data vydání jednotlivých verzí Ubuntu. In: *Ubuntu* [online]. London(Velká Británie): Canonical Group Limited [cit. 2016-02-29]. Dostupné z: https://wiki.ubuntu.com/Releases

[14] Nejbezpečnější operační systémy. In: *SecPoint* [online]. SecPoint [cit. 2016-02-29]. Dostupné z: https://www.secpoint.com/top-10-most-secure-operating-systems.html

[15] MySQL oficiální stránka. In: *MySQL* [online]. MySQL [cit. 2016-02-29]. Dostupné z: https://dev.mysql.com/doc/refman/5.7/en/what-is-mysql.html

[16] Nejpoužívanější databázové systémy. In: *DB-Engines* [online]. DB-Engines, 2016 [cit. 2016-02-29]. Dostupné z: http://db-engines.com/en/ranking

[17] Domovská stránka projektově. In: *Projektově.CZ s.r.o.* [online]. [cit. 2016-02-29]. Dostupné z: http://www.projektove.cz

[18] Domovská stránka Gitlab.com. In: *Gitlab* [online]. [cit. 2016-02-29]. Dostupné z: http://www.gitlab.com

[19] Bootstrap oficiální stránka. In: *Bootstrap* [online]. Bootstrap [cit. 2016-02-29]. Dostupné z: http://getbootstrap.com/about/

[20] O MVC. In: *W3schools* [online]. Refsnes Data [cit. 2016-03-01]. Dostupné z: http://www.w3schools.com/aspnet/mvc_intro.asp

[21] Active Records v Ruby on Rails. In: *RubyonRails.org* [online]. [cit. 2016-03-01]. Dostupné z: http://guides.rubyonrails.org/active_record_basics.html

[22] O vodopádovém modelu. In: *University of Missouri-St.Louis* [online]. [cit. 2016-03-01]. Dostupné z: http://www.umsl.edu/ hugheyd/is6840/waterfall.html

[23] Více informací o gemu Simple form. In: *GitHub* [online]. [cit. 2016-02-29]. Dostupné z: https://github.com/plataformatec/simple_form

[24] Více informací o gemu Devise. In: *GitHub* [online]. [cit. 2016-02-29]. Dostupné z: https://github.com/plataformatec/devise

[25] Více informací o gemu Haml. In: *GitHub* [online]. [cit. 2016-02-29]. Dostupné z: http://haml.info/

[26] Více informací o gemu Kaminari. In: *GitHub* [online]. [cit. 2016-02-29]. Dostupné z: https://github.com/amatsuda/kaminari

[27] Více informací o gemu RSpec. In: *GitHub* [online]. [cit. 2016-02-29]. Dostupné z: https://github.com/rspec/rspec-rail

[28] Kniha o testování v Ruby on Rails pomocí gemu RSpec. *SUMMERS, Aaron. Every Rails testing with RSpec. Leanpub, 2014.*

[29] Výuka jazyka HTML. In: *W3schools* [online]. Refsnes Data [cit. 2016-03-01]. Dostupné z: http://www.w3schools.com/html/html_intro.asp

[30] MVC. In: *Microsoft MSDN* [online]. Microsoft Corporation [cit. 2016-03-01]. Dostupné z: https://msdn.microsoft.com/en-us/library/ff649643.aspx

[31] Výuka jazyka SQL. In: *W3schools* [online]. Refsnes Data [cit. 2016-03-01]. Dostupné z: http://www.w3schools.com/sql/sql_intro.aspp

[32] Návod na práci se soubory s koncovkou ERB. In: *ApiDock* [online]. [cit. 2016-03-01]. Dostupné z: http://apidock.com/ruby/ERB

[33] Návod na práci se soubory s koncovkou XML. In: *W3schools* [online]. Refsnes Data [cit. 2016-03-01]. Dostupné z: http://www.w3schools.com/xml/